

# A high-performance Riccati based solver for tree-structured quadratic programs

**Frison, Gianluca; Kouzoupis, Dimitris; Diehl, Moritz; Jørgensen, John Bagterp**

*Published in:*
IFAC-PapersOnLine

*Link to article, DOI:*
10.1016/j.ifacol.2017.08.2027

*Publication date:*
2017

*Document Version*
Publisher's PDF, also known as Version of record

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 14399–14405

# A high-performance Riccati based solver for tree-structured quadratic programs ⋆

**Gianluca Frison** *,** **Dimitris Kouzoupis** ** **Moritz Diehl** **,***
**John Bagterp Jørgensen** *

* *Department of Applied Mathematics and Computer Science, Technical University of Denmark (e-mail: {giaf , jbjo} at dtu.dk)*
** *Department of Microsystems Engineering, University of Freiburg (e-mail: {dimitris.kouzoupis , moritz.diehl} at imtek.uni-freiburg.de)*
*** *Department of Mathematics, University of Freiburg, Germany*

**Abstract:** Robust multi-stage Model Predictive Control (MPC) is an increasingly popular approach to handle model uncertainties due to the simplicity of its problem formulation and other attractive properties. However, the exponential growth of the problem dimensions with respect to the robust horizon renders the online solution of such problems challenging and the development of tailored solvers crucial. In this paper, an interior point method is presented that can solve Quadratic Programs (QPs) arising in multi-stage MPC efficiently by means of a tree-structured Riccati recursion and a high-performance linear algebra library. A performance comparison with code-generated and general purpose sparse QP solvers shows that the computation times can be significantly reduced for all problem sizes that are practically relevant in embedded MPC applications. The presented implementation is freely available as part of the open-source software HPMPC.



*Keywords:* Predictive control, Quadratic programming, Tree structures, Numerical methods.

## 1. INTRODUCTION

Model Predictive Control (MPC) is an advanced control strategy that uses the model of the controlled plant to optimize its future behavior, subject to constraints (Rawlings and Mayne, 2009). However, when uncertainties are present in the model, the performance of the controller may deteriorate significantly (Grimm et al., 2004). Many approaches have been proposed in the literature to tackle this issue, such as min-max MPC (Campo and Morari, 1987) and tubed-based MPC (Mayne et al., 2011).

Another formulation that is recently gaining attention is the so-called (robust) multi-stage or scenario tree MPC, where the uncertainty is represented by a finite number of realizations at each decision point (de la Pena et al., 2005; Bernardini and Bemporad, 2009; Lucia et al., 2014). For the linear case, this approach leads to Quadratic Programs (QPs) with tree-structured dynamic constraints and dimensions that grow exponentially in the horizon length. As a result, structure-exploiting algorithms are essential for using multi-stage MPC in practical applications.

Several customized algorithms have been proposed in the literature to solve such tree-structured optimization problems. In the pioneering work of (Steinbach, 2002), an Interior Point (IP) method is used with a generic framework to recursively factorize the KKT matrix of a wide class of tree-structured optimization problems. This framework is closely related to the developments of this paper. Recent research focuses on very large problems, where distributed IP methods of different flavors have been developed (Blomvall and Lindberg, 2002; Pakazad et al., 2016; Hübner et al., 2016). The authors in (Sampathirao et al., 2015) implemented a dual accelerated proximal gradient method on a GPU to solve tree-structured convex problems in a massively parallelizable fashion.

A common practice to tackle scenario tree optimization problems is to decompose the tree into scenarios that are coupled to each other via constraints on the inputs, often referred to as non-anticipativity constraints. These constraints enforce that the control at each node of the tree is the same for all uncertainty realizations, i.e., the current realization of the uncertainty cannot be anticipated. This reformulation leads to optimization problems with simpler structure but higher dimension. Dual decomposition techniques have been successfully applied on such formulations that solve the dual problem using first-order methods (Marti et al., 2015) or Newton's method (Leidereiter et al., 2015; Klintberg et al., 2016).

In this paper, we directly tackle the lower dimensional tree-structured QP with an IP method as in Steinbach (2002), and employ a modified Riccati recursion to efficiently compute the search direction. We combine the algorithm with a tailored Linear Algebra (LA) library, especially suited for matrix dimensions that are common in most MPC problems. In contrast to existing software that focuses on large-scale problems, our aim is to provide high-performance solvers for small- to medium-scale embedded MPC ap-

⋆ Support by the EU via ERC-HIGHWIND (259 166), ITN-TEMPO (607 957), and ITN-AWESCO (642 682), by DFG via the Research Unit FOR 2401, by Ministerium für Wissenschaft, Forschung und Kunst Baden-Wuerttemberg (Az: 22-7533.-30-20/9/3), and by Det Frie Forskningsråd (DFF - 6111-00398) is gratefully acknowledged.


10.1016/j.ifacol.2017.08.2027

plications. We do not consider parallelization techniques, since we target sampling times in the millisecond range or below, where the overhead of parallelization makes its use of limited interest. By means of a numerical case study, we show that the resulting software, which is freely available online, can outperform general purpose sparse solvers as well as code-generated optimal control solvers designed for nominal MPC. Moreover, a comparison between nominal and robust multi-stage MPC with plant-model mismatch shows that the latter can guarantee constraint satisfaction with higher probability, without sacrificing significantly more computational resources.

The paper is organized as follows. Section 2 introduces the problem formulation. Section 3 presents the modified Riccati recursion for multi-stage MPC and describes how structure-exploitation can drastically reduce the computational load. Section 4 discusses software implementation details while Section 5 provides a detailed benchmarking of different methods using a simple numerical example. Section 6 concludes the paper.

## 2. PROBLEM FORMULATION

In this section, we introduce the notion of tree-structured dynamic constraints and derive the multi-stage MPC problem formulation.

### *2.1 Tree-structured dynamic constraints*

In multi-stage MPC, the system dynamics depend on an uncertain parameter $\theta$, which can take values from a finite set of realizations at each decision point [1] . With $k$ denoting the stage index, the linear (or linearized) dynamics take the form [2]

$$x_{k+1} = A_{k+1}(\theta)x_k + B_{k+1}(\theta)u_k + b_{k+1}(\theta), \qquad (1)$$

for $k = 0, \dots, N-1$ and $N$ the prediction horizon. Sampling the uncertainly in this fashion gives rise to a tree-structured system of equality constraints, since each realization in combination with the same control input will drive the system to a different state.

Let $m_d$ be the number of uncertainty realizations at each decision point. This sampling yields $m_d^N$ different scenarios that have to be considered in the problem formulation. To reduce the computational cost, it is a common practice to introduce the notion of a robust horizon length $N_r \leq N$ within which the dynamic constraints keep branching. The shorter the robust horizon, the smaller (but perhaps less robust) the optimization problem. An example of such tree with $m_d = 2, N_r = 2$ and $N = 3$ is depicted in Figure 1. We assume throughout the paper that the nodes, indexed by $n$, are numbered in some ascending order. This implies that for two nodes $m, n$ with $m < n$, it holds $k(m) \leq k(n)$ with $k(\cdot)$ denoting the stage index of a node.

In what follows, $\mathcal{N}$ is the set of the nodes in the tree and $\hat{N} := |\mathcal{N}|$ its cardinality, i.e., the total number of

[1] Uncertainty can also appear in the objective or in the constraints and the same approach is applicable.

[2] Note that the matrices and vectors of the system dynamics are indexed here with $k+1$ instead of the more common subscript $k$, to be consistent with the nodes of the tree as introduced later on.

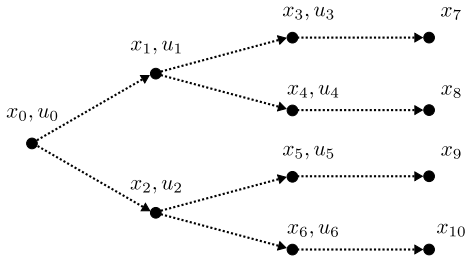


Fig. 1. An example of a tree with prediction horizon $N = 3$, robust horizon $N_r = 2$ and $m_d = 2$ realizations.

nodes. The set $\mathcal{L} := \{n \,|\, k(n) = N\}$ contains the leaves of the tree (with $|\mathcal{L}| = m_d^{N_r}$) while $\mathcal{C}(n)$ denotes the set of children of node $n$. Note that $|\mathcal{C}(n)| = m_d$ if $k(n) < N_r$. If $N_r \leq k(n) < N$, then $|\,\mathcal{C}(n)\,| = 1$ while $n \in \mathcal{L}$ implies that $|\,\mathcal{C}(n)\,| = 0$. For the example of Figure 1, $\mathcal{L} = \{7, 8, 9, 10\}$ and $\mathcal{C}(1) = \{3, 4\}$.

### *2.2 Multi-stage MPC*

Using the notation of the previous section, the node-wise separable objective function with tree-structured dynamic constraints and bounds on states and controls can be summarized as

$$\begin{aligned}
\min_{x,u} \quad & \sum_{n \in \mathcal{N}\backslash\mathcal{L}} \frac{1}{2} \begin{bmatrix} x_n \\ u_n \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_n & S_n^T & q_n \\ S_n & R_n & r_n \\ q_n^T & r_n^T & 0 \end{bmatrix} \begin{bmatrix} x_n \\ u_n \\ 1 \end{bmatrix} + \\
& + \sum_{n \in \mathcal{L}} \frac{1}{2} \begin{bmatrix} x_N \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_N & q_N \\ q_N^T & 0 \end{bmatrix} \begin{bmatrix} x_N \\ 1 \end{bmatrix} \\
\text{s.t.} \quad & x_m = A_m x_n + B_m u_n + b_m, \quad n \in \mathcal{N}\backslash\mathcal{L},\, m \in \mathcal{C}(n), \\
& u_n^{\mathrm{l}} \leq u_n \leq u_n^{\mathrm{u}}, \quad n \in \mathcal{N}\backslash\mathcal{L}, \\
& x_n^{\mathrm{l}} \leq x_n \leq x_n^{\mathrm{u}}, \quad n \in \mathcal{N},
\end{aligned}$$

where $x_n \in \mathbb{R}^{n_x}$ are the differential states and $u_n \in \mathbb{R}^{n_u}$ the control inputs of node $n$. The vectors $x, u$ comprise the concatenated state and input trajectories respectively. Note that the quadratic stage cost of each node typically contains a weight associated with the probability that this node occurs or simply a scaling factor so as to minimize the average cost over all scenarios. Note also that the matrices of the system dynamics are associated to the children nodes, as in (1).

*Remark 1.* The algorithm and software presented in this paper can handle general polyhedral constraints on states and controls. The restriction to bounds is only made here for notational convenience.

## 3. A MODIFIED RICCATI RECURSION

This section contains the derivation of a backward Riccati recursion algorithm tailored to tree-structured unconstrained MPC problems. This recursion could be reduced to the generic framework in (Steinbach, 2002); however, a tailored derivation makes the presentation of the algorithm more clear.

It is well known that the backward Riccati recursion can be employed to factorize stage-wise the KKT matrix of the unconstrained nominal MPC problem. Therefore, it is often used to compute the search direction in IP methods for nominal MPC problems (Rao et al., 1998). The nominal

MPC problem can be seen as a multi-stage MPC problem where all nodes have at most one child. This implies that the notion of stage and node are essentially equivalent and the Riccati recursion can be seen as a factorization where the tree is processed node-wise.

In the more general case where a node can have more than one children, the notion of node and stage can not be interchanged any longer. It is however still possible to derive a modified factorization procedure that processes the tree node-wise, taking into account information from all children nodes.

### *3.1 Recursion for nodes with one child*

This is the standard Riccati recursion case, except for the different indexing in the matrices and vectors of the dynamic equations.

One stage of the KKT matrix with the corresponding terminal cost looks like

$$\left[\begin{array}{ccc|c} Q_n & S_n^T & A_m^T & \\ S_n & R_n & B_m^T & \\ A_m & B_m & & -I \\ \hline & & -I & P_m \end{array}\right] \left[\begin{array}{c} x_n \\ u_n \\ \pi_m \\ \hline x_m \end{array}\right] = \left[\begin{array}{c} q_n \\ r_n \\ b_m \\ \hline p_m \end{array}\right], \tag{2}$$

where in this case $\mathcal{C}(n) = \{m\}$.

The variable $x_m$ can be eliminated by adding the last block-row to the third block-row multiplied by $P_m$,

$$\begin{bmatrix} Q_n & S_n^T & A_m^T \\ S_n & R_n & B_m^T \\ P_m A_m & P_m B_m & -I \end{bmatrix} \begin{bmatrix} x_n \\ u_n \\ \pi_m \end{bmatrix} = \begin{bmatrix} q_n \\ r_n \\ P_m b_m + p_m \end{bmatrix}.$$

The variable $\pi_m$ can be eliminated by adding the last block-row multiplied by $A_m^T$ to the first block-row, and by adding the last block-row multiplied by $B_m^T$ to the second block-row,

$$\begin{bmatrix} \widetilde{Q}_n & \widetilde{S}_n^T \\ \widetilde{S}_n & \widetilde{R}_n \end{bmatrix} \begin{bmatrix} x_n \\ u_n \end{bmatrix} = \begin{bmatrix} \widetilde{q}_n \\ \widetilde{r}_n \end{bmatrix},$$

where the updated matrices and vectors are

$$\widetilde{Q}_n = Q_n + A_m^T P_m A_m \tag{3a}$$
$$\widetilde{S}_n = S_n + B_m^T P_m A_m \tag{3b}$$
$$\widetilde{R}_n = R_n + B_m^T P_m B_m \tag{3c}$$
$$\widetilde{q}_n = q_n + A_m^T (P_m b_m + p_m) \tag{3d}$$
$$\widetilde{r}_n = r_n + B_m^T (P_m b_m + p_m). \tag{3e}$$

Finally, the variable $u_n$ can be eliminated adding the last block-row multiplied by $-\widetilde{S}_n^T \widetilde{R}_n^{-1}$ to the first block-row, obtaining

$$[P_n]\,[x_n] = [p_n]$$

where

$$P_n = \widetilde{Q}_n - \widetilde{S}_n^T \widetilde{R}_n^{-1} \widetilde{S}_n \tag{4}$$

and

$$p_n = \widetilde{q}_n - \widetilde{S}_n^T \widetilde{R}_n^{-1} \widetilde{r}_n. \tag{5}$$

This concludes the calculations for nodes with one child.

### *3.2 Recursion for nodes with two or more children*

For the shake of simplicity, let us consider the case of only two children per node.

In the tree case, the index of matrices and vectors is associated to the node, and not the stage as in the standard Riccati recursion. Let the indexes $m_1$ and $m_2$ refer to the two elements of $\mathcal{C}(n)$, i.e., $\mathcal{C}(n) = \{m_1, m_2\}$. One stage of the KKT matrix with the corresponding terminal cost looks like

$$\left[\begin{array}{cccc|cc} Q_n & S_n^T & A_{m_1}^T & A_{m_2}^T & & \\ S_n & R_n & B_{m_1}^T & B_{m_2}^T & & \\ A_{m_1} & B_{m_1} & & & -I & \\ A_{m_2} & B_{m_2} & & & & -I \\ \hline & & -I & & P_{m_1} & \\ & & & -I & & P_{m_2} \end{array}\right] \left[\begin{array}{c} x_n \\ u_n \\ \pi_{m_1} \\ \pi_{m_2} \\ \hline x_{m_1} \\ x_{m_2} \end{array}\right] = \left[\begin{array}{c} q_n \\ r_n \\ b_{m_1} \\ b_{m_2} \\ \hline p_{m_1} \\ p_{m_2} \end{array}\right]$$

Note that this looks identical to (2), where

$$x_m = \begin{bmatrix} x_{m_1} \\ x_{m_2} \end{bmatrix},\ A_m = \begin{bmatrix} A_{m_1} \\ A_{m_2} \end{bmatrix},\ B_m = \begin{bmatrix} B_{m_1} \\ B_{m_2} \end{bmatrix},\ b_m = \begin{bmatrix} b_{m_1} \\ b_{m_2} \end{bmatrix}$$
$$P_m = \begin{bmatrix} P_{m_1} & \\ & P_{m_2} \end{bmatrix},\ p_m = \begin{bmatrix} p_{m_1} \\ p_{m_2} \end{bmatrix},\ \pi_m = \begin{bmatrix} \pi_{m_1} \\ \pi_{m_2} \end{bmatrix}$$

i.e. the two children nodes are considered as a single stage with increased variable sizes.

This suggests that it is possible to use the standard Riccati recursion to handle trees, at the cost of increasing the variable sizes and ignoring the specific structure of the problem.

Due to the block-diagonal structure of the recursion matrix $P_m$ in the tree case, the matrices and vectors in (3) look like

$$\widetilde{Q}_n = Q_n + A_{m_1}^T P_{m_1} A_{m_1} + A_{m_2}^T P_{m_2} A_{m_2}$$
$$\widetilde{S}_n = S_n + B_{m_1}^T P_{m_1} A_{m_1} + B_{m_2}^T P_{m_2} A_{m_2}$$
$$\widetilde{R}_n = R_n + B_{m_1}^T P_{m_1} B_{m_1} + B_{m_2}^T P_{m_2} B_{m_2}$$
$$\widetilde{q}_n = q_n + A_{m_1}^T (P_{m_1} b_{m_1} + p_{m_1}) + A_{m_2}^T (P_{m_2} b_{m_2} + p_{m_2})$$
$$\widetilde{r}_n = r_n + B_{m_1}^T (P_{m_1} b_{m_1} + p_{m_1}) + B_{m_2}^T (P_{m_2} b_{m_2} + p_{m_2}),$$

where the updates from the different children are added together.

Once the updated matrices and vectors have been computed, the recursion matrix $P_n$ and vector $p_n$ are computed as in (4) and (5).

### *3.3 Computational cost analysis*

The algorithm for the factorization of the KKT matrix of the tree MPC problem is presented in Algorithm 1. The

**Algorithm 1** Tree Riccati - backward factorization

**for** each node $n$ in $\hat{N}$ to 0 **do**
  **if** $n \in \mathcal{L}$ **then**
    $\mathcal{L}_n \leftarrow Q_n^{1/2}$
  **else**
    $\begin{bmatrix} \widetilde{R}_n \\ \widetilde{S}_n^T \ \widetilde{Q}_n \end{bmatrix} \leftarrow \begin{bmatrix} R_n \\ S_n^T \ Q_n \end{bmatrix}$
    **for** each child m **do**
      $D \leftarrow \begin{bmatrix} B_m^T \\ A_m^T \end{bmatrix} \mathcal{L}_m$
      $\begin{bmatrix} \widetilde{R}_n \\ \widetilde{S}_n^T \ \widetilde{Q}_n \end{bmatrix} \leftarrow \begin{bmatrix} \widetilde{R}_n \\ \widetilde{S}_n^T \ \widetilde{Q}_n \end{bmatrix} + DD^T$
    **end for**
    $\begin{bmatrix} \Lambda_n \\ L_n \ \mathcal{L}_n \end{bmatrix} \leftarrow \begin{bmatrix} \widetilde{R}_n \\ \widetilde{S}_n^T \ \widetilde{Q}_n \end{bmatrix}^{1/2}$
  **end if**
**end for**

**Algorithm 2** Tree Riccati - backward substitution

**for** each node $n$ in $\hat{N}$ to 0 **do**
  **if** $n \in \mathcal{L}$ **then**
    $p_n \leftarrow q_n$
  **else**
    $\begin{bmatrix}\widetilde{r}_n \\ \widetilde{q}_n\end{bmatrix} \leftarrow \begin{bmatrix}r_n \\ q_n\end{bmatrix}$
    **for** each child $m$ **do**
      $d \leftarrow \mathcal{L}_m \mathcal{L}_m^T b_m + p_m$
      $\begin{bmatrix}\widetilde{r}_n \\ \widetilde{q}_n\end{bmatrix} \leftarrow \begin{bmatrix}\widetilde{r}_n \\ \widetilde{q}_n\end{bmatrix} + \begin{bmatrix}B_m^T \\ A_m^T\end{bmatrix} d$
    **end for**
    $l_n \leftarrow \Lambda_n^{-1} \widetilde{r}_n$
    $p_n \leftarrow \widetilde{q}_n - L_n l_n$
  **end if**
**end for**

**Algorithm 3** Tree Riccati - forward substitution

**for** each node $n$ in 0 to $\hat{N}$ **do**
  **if** $n \notin \mathcal{L}$ **then**
    $u_n \leftarrow -\Lambda_n^{-T}(l_n + L_n^T x_n)$
    **for** each child $m$ **do**
      $x_m \leftarrow A_m x_n + B_m u_n + b_m$
      $\pi_m \leftarrow \mathcal{L}_m \mathcal{L}_m^T x_m + p_m$
    **end for**
  **end if**
**end for**

algorithms for the forward and backward substitution are presented in Algorithms 2 and 3.

Assuming that all stages have the same state and input vector dimensions $n_x$ and $n_u$, Algorithm 1 has a computational cost per node of

$$C_{m_d}(n_x, n_u) = m_d \left(2n_x^3 + 3n_x^2 n_u + n_x n_u^2\right) + \tfrac{1}{3}(n_x + n_u)^3$$

flops for $n \in \mathcal{N} \backslash \mathcal{L}$, and

$$C_f(n_x) = \tfrac{1}{3} n_x^3$$

flops for $n \in \mathcal{L}$. Therefore, the cost per node does not depend on $N_r$ or $N$, and it is less than a factor $m_d$ compared to the standard Riccati recursion for the nominal MPC problem.

For a tree with $m_d$ realizations, robust horizon $N_r$ and prediction horizon $N$, the computational cost for the KKT matrix factorization is of

$$\frac{{m_d}^{N_r+1} - 1}{m_d - 1} C_{m_d}(n_x, n_u) + \\ + {m_d}^{N_r} \left((N - N_r) C_1(n_x, n_u) + C_f(n_x)\right)$$

flops, where

$$C_1(n_x, n_u) = \tfrac{7}{3} n_x^3 + 4 n_x^2 n_u + 2 n_x n_u^2 + \tfrac{1}{3} n_u^3$$

flops is the cost per node in case there is only one child (that is equal to the cost per stage of the standard Riccati recursion for the nominal MPC problem). Therefore, the factorization cost is less than $m_d$ times the standard Riccati recursion cost for an horizon equal to the number of nodes $\hat{N}$, and it scales linearly with the number of scenarios ${m_d}^{N_r}$.

In case the standard Riccati recursion is employed to solve the tree MPC problem, the state and input vector dimensions increase at each branch in the tree. The computational cost for the KKT matrix factorization is

$$\sum_{k=0}^{N_r} C_{m_d}({m_d}^k n_x, {m_d}^k n_u) + \\ + (N - N_r) C_1({m_d}^{N_r} n_x, {m_d}^{N_r} n_u) + C_f({m_d}^{N_r} n_x)$$

flops, that scales cubically in the number of scenarios ${m_d}^{N_r}$ (and therefore it is approximately ${m_d}^{2N_r}$ times larger than using the tree Riccati recursion), since $C_{m_d}$, $C_1$ and $C_f$ are cubic in $n_x$ and $n_u$.

## 4. EMBEDDING IN THE HPMPC FRAMEWORK

This section discusses implementation details of the presented tree-structured IP method.

### *4.1 Linear algebra*

The implementation of the tree Riccati algorithm requires the same LA routines as the standard Riccati recursion. Algorithms 1, 2 and 3 have been implemented using the high-performance LA routines in BLASFEO (Frison et al., 2017). BLASFEO (which stands for BLAS For Embedded Optimization) is a LA library providing routines tailored for small-scale matrices and optimized for a number of computer architecures.

Compared to the LA routines present in HPMPC, BLASFEO provides a new interface, that defines matrix and vector structure types to completely hide any implementation detail to the user (e.g. the programmer coding the MPC algorithm). In particular, two optimized implementations are provided at the moment (besides a reference implementation): one based on BLAS (internally making use of the column-major matrix storage), and one based on custom LA optimized for embedded optimization (internally making use of the panel-major matrix format) (Frison, 2015).

This allows for the same algorithm to be coded once, and then linked to different versions of BLASFEO.

### *4.2 IP method routines*

The HPMPC library (Frison et al., 2014; HPMPC, 2014) contains a high-performance implementation of a Mehrotra's predictor-corrector IP method tailored to MPC problems, employing a backward Riccati recusion to compute the search direction. The key routines in an IP method are the factorization and solution of the KKT matrix of the equality constrained sub-problems, accounting for all the terms cubic and quadratic in $n_x$ and $n_u$ in the computational complexity. Nevertheless, in the case of small-scale QPs, typical of MPC applications, all remaining auxiliary routines (e.g. computation of the update term of Hessian and gradient, computation of the step length, update of the variables) have also a non-negligible cost. Therefore, highly-optimized versions of all these routines are also provided in HPMPC.

The IP method in HPMPC is implemented such that the number of states and inputs can change stage-wise. This feature allows for the use of the same auxiliary routines in both nominal and robust cases.

## 5. NUMERICAL EXPERIMENTS

In this section, we compare the developed structure-exploiting IP method (Tree HPMPC) against state-of-the-art numerical solvers. The benchmark example is a linear MPC controller for regulating a chain of oscillating masses.

### 5.1 Problem description

The test problem with the chain of masses and springs is described in (Wang and Boyd, 2010). It is a linear system where the number of masses $m$ (and consequently the number of states $n_x = 2m$) can be easily scaled. In all our tests, a force acts on each of the first $m-1$ masses, i.e., the number of control inputs is $n_u = m-1$. The nominal value for the ratio between the spring constant and the mass value is $r = 1$, but it can be changed to introduce model-plant mismatch. The continuous time system is sampled with a period $T_s = 0.2$ s. The weights on the cost function are equal to $R = I$ for the inputs and $Q = 10I$ for the states. There are bounds on all inputs and states components, equal to $\pm 1$ for the inputs, $\pm 1$ for the displacement and $\pm 2$ for the velocities of the masses.

### 5.2 Feasibility under plant-model mismatch

Let us consider the case of $m = 4$ masses and a prediction horizon of $N = 10$ steps to introduce the benchmark example and assess the performance of nominal and robust multi-stage MPC under plant-model mismatch. We excite the system with velocities $v_3 = -1.7$ and $v_4 = 1.2$ on the last two masses and run closed-loop simulations with the ratio of the controlled plant first equal to $r = 1$ and then $r = 0.8$. The nominal MPC controller assumes $r = 1$ while the robust formulation uses $N_r = 2$ and $m_d = 3$ with $r(\theta) \in \{0.8, 1, 1.2\}$. As depicted in Figure 2, the nominal MPC scheme fails to respect the constraints on the position of the masses,[3] as opposed to multi-stage MPC that retains feasibility. Note that all three realizations are weighted with the same probability in the scenario tree and therefore the controller optimizes the average performance over all scenarios.

### 5.3 Standard versus tree Riccati recursion

Here we compare the time needed to factorize the KKT matrix for the unconstrained tree MPC problem, by means of a *Standard* Riccati recursion (S.R.) and a *Tree* Riccati recursion (T.R.). Two test problems are considered, one small with $m = 2$ masses (and therefore $n_x = 2m = 4$ states and $n_u = m - 1 = 1$ control), and one large with $m = 8$ masses (and therefore $n_x = 2m = 16$ states and $n_u = m - 1 = 7$ controls). The prediction horizon is fixed to $N = 5$, while several values for the robust horizon $N_r$ and the number of realizations $m_d$ are considered. The LA used in the Riccati algorithms is provided by either the *high-performance* (HP) implementation of BLASFEO or by *OpenBLAS* (OB), which is the best performing open-source BLAS implementation, highly optimized for many computer architectures (OpenBLAS, 2011). The latter library is called through the common interface described in Section 4.1. The results are reported in Table 1.

[3] Infeasible problems are treated by re-applying the previous controller input.

Table 1. Timings ($\mu$s) for the Riccati factorization of the KKT matrix of the unconstrained MPC problem, for $m = 2$ masses and $m = 8$ masses. Problem size $N = 5$, $n_x = 2m$, $n_u = m - 1$. Test processor: Intel Core i7 4800MQ (Haswell architecture).

| | | | $N_r = 1$ | | | $N_r = 2$ | |
|---|---|---|---|---|---|---|---|
| Method | $m$ | LA | $m_d = 1$ | $m_d = 2$ | $m_d = 3$ | $m_d = 2$ | $m_d = 3$ |
| S.R. | 2 | OB | 2.43 | 5.63 | 10.6 | 13.3 | 82.9 |
| S.R. | 2 | HP | 0.94 | 2.40 | 3.59 | 4.81 | 24.9 |
| T.R. | 2 | OB | 2.50 | 4.98 | 7.64 | 8.91 | 19.3 |
| T.R. | 2 | HP | 1.19 | 2.27 | 3.37 | 4.10 | 8.92 |
| S.R. | 8 | OB | 20.0 | 95.5 | 218 | 356 | 2564 |
| S.R. | 8 | HP | 7.10 | 27.8 | 75.8 | 129 | 1202 |
| T.R. | 8 | OB | 20.2 | 40.7 | 60.4 | 74.4 | 164 |
| T.R. | 8 | HP | 7.16 | 14.4 | 21.5 | 26.4 | 58.6 |

For the small problem and a number of scenarios $m_d^{N_r}$ up to 4, the speed-up obtained using the Tree Riccati algorithm is negligible. This is due to the fact that well-optimized LA routines generally have poor performance for very small matrices, while it increases quickly for matrices of size up to a few tens. Therefore, the additional number of flops is hidden by the increased LA performance.

For the larger problem with more scenarios the speed-up given by the Tree Riccati becomes much larger. In the case of $N_r = 2$ and $m_d = 3$ (giving $3^2 = 9$ scenarios) the speed-up is of about 15-20 times depending on the LA.

Finally, the use of the high-performance implementation of BLASFEO provides a speed-up of about 2-3 times (depending on the problem size) with respect to the use of OpenBLAS, which is better suited for large-scale matrices. Therefore, BLASFEO will provide the LA for all remaining numerical experiments. For the large problem with $N_r = 2$ and $m_d = 3$, the combination of tree Riccati recursion and BLASFEO gives a speed-up of more than 40 times with respect to standard Riccati recursion and OpenBLAS.

### 5.4 Performance of Tree HPMPC

Aim of this section is to compare the performance of Tree HPMPC against existing software for different problem sizes. Both code-generated and general sparse solvers are considered in this benchmark. The selected solvers are:

- CVXGEN: a code generator for small- to medium-scale convex optimization problems (roughly up to 4000 non-zero entries in the KKT matrix) (Mattingley and Boyd, 2012).
- OOQP: A primal-dual IP method for convex QPs, written in C++ (Gertz and Wright, 2003). The software supports a variety of structured QPs as well as general sparse QPs. The latter functionality is used here since there is currently no module that supports tree-structured constraints.
- FORCES Pro: a code generator of IP methods tailored to multi-stage QPs (Domahidi et al., 2012).
- HPMPC: The already introduced IP method both with a standard and a tree Riccati recursion.

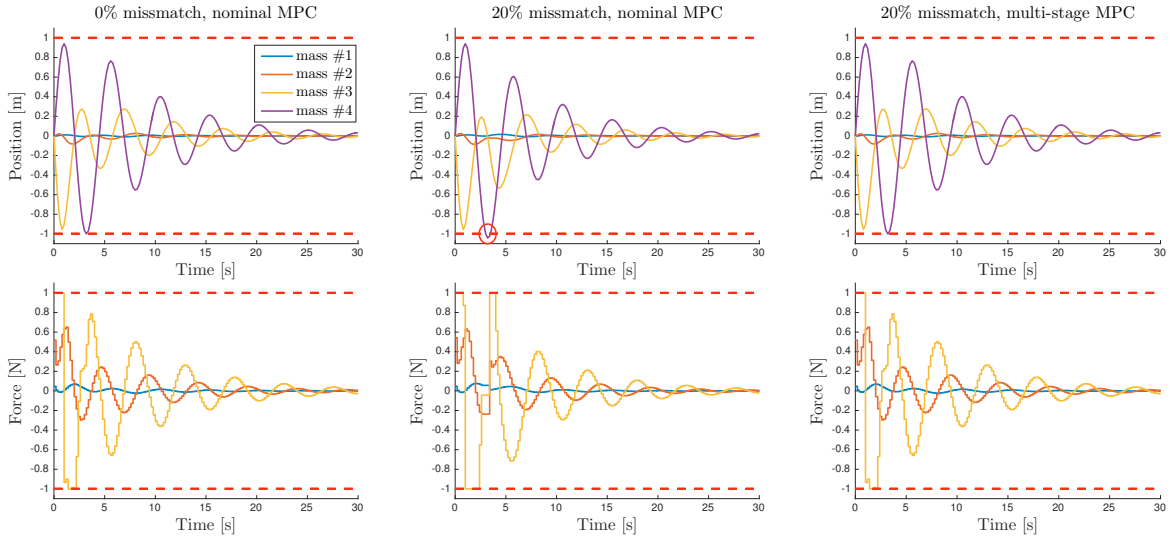


Fig. 2. Closed loop trajectories of positions and controls for nominal MPC without plant-model mismatch (left), nominal MPC with plant-model mismatch (middle) and robust multi-stage MPC with plant-model mismatch (right).

Table 2. Number of variables and average/maximum timings (ms) for two benchmark problems. First problem with $m = 2$ and $N = 5$. Second problem with $m = 4$ and $N = 10$. Test processor: Intel Xeon E5-2687W (Sandy Bridge architecture).

| Tree parameters | $N_r = 1$ | | | $N_r = 2$ | | $N_r = 3$ | |
|---|---|---|---|---|---|---|---|
| | $m_d = 1$ | $m_d = 2$ | $m_d = 3$ | $m_d = 2$ | $m_d = 3$ | $m_d = 2$ | $m_d = 3$ |
| Solvers / # variables | 29 | 54 | 79 | 91 | 192 | 146 | 443 |
| CVXGEN | 0.07/0.08 | 0.17/0.24 | 0.28/0.39 | 0.36/0.55 | 0.77/1.36 | 0.64/0.85 | -/- |
| OOQP | 0.49/0.65 | 0.78/1.08 | 1.12/1.53 | 1.33/2.02 | 2.92/4.45 | 2.21/4.29 | 7.82/13.03 |
| FORCES Pro | 0.07/0.10 | 0.17/0.21 | 0.34/0.45 | 0.51/0.64 | 4.02/5.56 | 1.52/2.19 | 56.35/73.00 |
| Standard HPMPC | 0.04/0.05 | 0.06/0.10 | 0.09/0.13 | 0.12/0.19 | 0.42/0.57 | 0.27/0.49 | 3.47/4.69 |
| Tree HPMPC | 0.04/0.06 | 0.08/0.11 | 0.13/0.21 | 0.15/0.22 | 0.32/0.48 | 0.24/0.34 | 0.61/0.82 |
| Solvers / # variables | 118 | 128 | 338 | 417 | 911 | 754 | 2438 |
| CVXGEN | 0.71/0.91 | 1.38/1.73 | -/- | -/- | -/- | -/- | -/- |
| OOQP | 2.28/3.33 | 4.78/8.50 | 7.66/12.57 | 9.74/15.61 | 24.07/39.98 | 19.50/33.80 | 80.52/104.65 |
| FORCES Pro | 0.56/0.69 | 1.22/1.57 | 2.93/3.76 | 6.32/8.65 | 98.37/132.19 | 48.82/65.49 | 2617.19/3933.07 |
| Standard HPMPC | 0.12/0.19 | 0.29/0.44 | 0.61/0.87 | 0.88/1.38 | 5.48/7.41 | 3.64/5.12 | 105.98/142.58 |
| Tree HPMPC | 0.14/0.21 | 0.28/0.40 | 0.43/0.60 | 0.52/0.77 | 1.14/1.60 | 0.90/1.24 | 3.13/4.22 |

The above solvers are used in closed-loop simulations where the linear MPC controller regulates the system to the origin after an initial perturbation. Average and maximum timings for a small ($N = 5, m = 2$) and a large ($N = 10, m = 4$) problem are reported in Table 2 (all timings are performed in C code). No plant-model mismatch is considered here ($r = 1$) to avoid infeasible problems for the nominal MPC schemes. Note that although the same problem formulation that is passed to Standard HPMPC can be also used within FORCES Pro, the recently developed tool Y2F (Y2F, 2016) is used instead to automatically translate the problem from YALMIP (Löfberg, 2004) to the expected multi-stage form. This approach seemed to be both simpler and more efficient for the conducted experiments. The missing entries of CVXGEN in the table indicate problem instances where code generation failed due to large number of variables.

From Table 2 it is clear that the solvers for nominal MPC (Standard HPMPC) and multi-stage QPs (FORCES Pro) perform reasonably well for a small number of scenarios, but their computational cost explodes as the number of scenarios increases. Standard HPMPC scales better due to the superior performance of the LA routines in BLASFEO. On the other hand, solvers for generic sparse QPs (CVXGEN and OOQP) are able to exploit the problem structure and their performance scales well with the number of scenarios. CVXGEN is roughly 4-8 times faster than OOQP for small problems where the code generation succeeds.

Tree HPMPC is the fastest solver in most cases (and for the larger problems by a wide margin), since the tree Riccati recursion can exploit well the tree-structure of the QPs. It is interesting to note that the ratio of the computational time between OOQP and Tree HPMPC is

rather constant with respect to the number of scenarios, and equal to about 10 for the small problem and 20 for the large problem. This is a clear hint to the fact that both solvers are equally able to exploit the sparsity in the problem, with the difference in computational time coming from the much higher performance of the dense LA routines in BLASFEO compared to the sparse LA employed in OOQP.

## 6. CONCLUSION

In this paper, we presented a tailored Riccati recursion algorithm to efficiently factorize the KKT matrix of unconstrained, tree-structured QPs arising in multi-stage MPC. The computational cost of the method scales linearly in the number of nodes in the tree describing the MPC problem. Furthermore, the computational cost per node is smaller than the computational cost in the standard Riccati recursion case times the number of children of the node. The tree Riccati recursion has been implemented using the high-performance LA routines in BLASFEO and embedded into a Mehrotra's type IP method. The developed solver compares favorably to several existing alternatives, showing that the combination of a structure-exploiting algorithm with high-performance dense LA routines tailored for small-scale matrices can outperform by a wide margin code-generated solvers as well as general purpose sparse solvers for this class of problems.

## REFERENCES


Bernardini, D. and Bemporad, A. (2009). Scenario-based model predictive control of stochastic constrained linear systems. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 6333–6338.

Blomvall, J. and Lindberg, P.O. (2002). A riccati-based primal interior point solver for multistage stochastic programming. *European Journal of Operational Research*, 143, 452–461.

Campo, P.J. and Morari, M. (1987). Robust model predictive control. In *Proceedings of the American Control Conference (ACC)*.

de la Pena, D.M., Bemporad, A., and Alamo, T. (2005). Stochastic programming applied to model predictive control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Domahidi, A., Zgraggen, A., Zeilinger, M., Morari, M., and Jones, C. (2012). Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 668–674. Maui, HI, USA.

Frison, G., Sorensen, H.B., Dammann, B., and Jørgensen, J.B. (2014). High-performance small-scale solvers for linear model predictive control. In *Proceedings of the European Control Conference (ECC)*, 128–133.

Frison, G. (2015). *Algorithms and Methods for High-Performance Model Predictive Control*. Ph.D. thesis, Technical University of Denmark (DTU).

Frison, G., Kouzoupis, D., Zanelli, A., and Diehl, M. (2017). BLASFEO: Basic linear algebra subroutines for embedded optimization. *arXiv preprint*.

Gertz, E.M. and Wright, S.J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1), 58–81.

Grimm, G., Messina, M., Tuna, S., and Teel, A. (2004). Examples when nonlinear model predictive control is nonrobust. *Automatica*, 40, 1729–1738.

HPMPC (2014). HPMPC: High-performance MPC. https://github.com/giaf/hpmpc.

Hübner, J., Schmidt, M., and Steinbach, M.C. (2016). A distributed interior-point KKT solver for multistage stochastic optimization. *Optimization Online*.

Klintberg, E., Dahl, J., Fredriksson, J., and Gros, S. (2016). An improved dual Newton strategy for scenario-tree MPC. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Leidereiter, C., Potschka, A., and Bock, H.G. (2015). Dual decomposition for QPs in scenario tree NMPC. In *Proceedings of the European Control Conference (ECC)*.

Löfberg, J. (2004). YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*.

Lucia, S., Andersson, J.A.E., Brandt, H., Diehl, M., and Engell, S. (2014). Handling uncertainty in economic nonlinear model predictive control: A comparative case study. *Jounal of Process Control*, 24, 1247–1259.

Marti, R., Lucia, S., Sarabia, D., Paulen, R., Engell, S., and de Prada, C. (2015). An efficient distributed algorithm for multi-stage robust nonlinear predictive control. In *Proceedings of the European Control Conference (ECC)*.

Mattingley, J. and Boyd, S. (2012). CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 1–27.

Mayne, D., Kerrigan, E., van Wyk, E.J., and Falugi, P. (2011). Tube-based robust nonlinear model predictive control. *International Journal of Robust and Nonlinear Control*, 21, 1341–1353.

OpenBLAS (2011). OpenBLAS: An optimized BLAS library. http://www.openblas.net/.

Pakazad, S.K., Hansson, A., Andersen, M.S., and Nielsen, I. (2016). Distributed primal-dual interior-point methods for solving tree-structured coupled convex problems using message-passing. *Optimization Methods & Software*.

Rao, C., Wright, S., and Rawlings, J. (1998). Application of Interior-Point Methods to Model Predictive Control. *Journal of Optimization Theory and Applications*, 99, 723–757.

Rawlings, J. and Mayne, D. (2009). *Model Predictive Control: Theory and Design*. Nob Hill.

Sampathirao, A.K., Sopasakis, P., Bemporad, A., and Patrinos, P. (2015). Distributed solution of stochastic optimal control problems on GPUs. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Steinbach, M.C. (2002). Tree-sparse convex programs. *Mathematical Methods of Operations Research*, 56(3), 347–376.

Wang, Y. and Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2), 267–278.

Y2F (2016). Y2F: YALMIP to FORCES Pro interface. https://github.com/embotech/Y2F.